Optim Eng
DOI 10.1007/s11081-016-9343-0

A generic framework for handling constraints with agent-based optimization algorithms and application to aerodynamic design

Daniel J. Poole[1] · **Christian B. Allen**[1] · **Thomas C. S. Rendall**[1]

Received: 21 January 2016 / Revised: 20 June 2016 / Accepted: 14 October 2016

Abstract A generic constraint handling framework for use with any swarm-based optimization algorithm is presented. For swarm optimizers to solve constrained optimization problems effectively modifications have to be made to the optimizers to handle the constraints, however, these constraint handling frameworks are often not universally applicable to all swarm algorithms. A constraint handling framework is therefore presented in this paper that is compatible with any swarm optimizer, such that a user can wrap it around a chosen swarm algorithm and perform constrained optimization. The method, called separation-sub-swarm, works by dividing the population based on the feasibility of individual agents. This allows all feasible agents to move by existing swarm optimizer algorithms, hence promoting good performance and convergence characteristics of individual swarm algorithms. The framework is tested on a suite of analytical test function and a number of engineering benchmark problems, and compared to other generic constraint handling frameworks using four different swarm optimizers; particle swarm, gravitational search, a hybrid algorithm and differential evolution. It is shown that the new framework produces superior results compared to the established frameworks for all four swarm algorithms tested. Finally, the framework is applied to an aerodynamic shape optimization design problem where a shock-free solution is obtained.

Keywords Optimization · Constraint handling · Swarm algorithms · Aerodynamic shape optimization

✉ Daniel J. Poole
d.j.poole@bristol.ac.uk

Christian B. Allen
c.b.allen@bristol.ac.uk

Thomas C. S. Rendall
thomas.rendall@bristol.ac.uk

[1] Department of Aerospace Engineering, University of Bristol, Bristol BS8 1TR, UK

Published online: 24 November 2016

1 Introduction

Optimization is the process of improving on a current solution. In engineering design, historically, optimization has often been performed manually where designers use intuition to produce solutions to problems so that the solution performs better than the initial starting point. However, it has now become commonplace to couple numerical optimization algorithms with computational analysis to provide a robust engineering optimization approach and an example of this is the coupling of iterative computational fluid dynamics (CFD) methods to optimization algorithms to produce an aerodynamic shape optimization (ASO) process (Jameson et al. 1998; Martins et al. 2004; Hicken and Zingg 2010; Allen and Rendall 2013). The solution to many engineering optimization problems requires feasibility; constraints appear on the total cost, or other physical barriers to the solution, that must be adhered to. Mathematically, this can be described by statement 1; a single objective constrained optimization problem.

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbf{S}\in\Re^{\mathfrak{D}}} f(\mathbf{x}) \\ &\text{subject to} \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \\ &\mathbf{h}(\mathbf{x}) = \mathbf{0} \end{aligned} \tag{1}$$

In statement 1, $\mathbf{x}$ is the solution vector $[x^1, x^2, \ldots, x^D]^T$ where each element of the vector is a design variable; $f(\mathbf{x})$ is the value of the objective function for the given solution vector; $\mathbf{g}(\mathbf{x})$ represents inequality constraints and $\mathbf{h}(\mathbf{x})$ represents equality constraints; $\mathbf{S}$ is the bounded region of $\Re^{\mathfrak{D}}$ where the solution must lie, which has an upper bound in the kth dimension, U^k, and a lower bound, L^k, where $k = \{1, 2, \ldots, D\}$.

The choice of optimization algorithm for solving statement 1 is often driven by the degree of multimodality present in the problem, where global search algorithms are popular for multimodal problems. Many global search algorithms are agent-based, so use a set of agents who evolve and move by various mechanisms (often inspired by nature) to provide robust design space interrogation. However, to handle constraints, often an ad-hoc method is used that is added onto the optimization algorithm which usually either alters the optimization problem or the algorithm itself. Many of the state-of-the-art constraint handling methods are incompatible with all agent-based algorithms so it would be an advantage for a constraint handling method to be applicable for use with any agent-based optimization algorithm. A user of the framework could then wrap it around any swarm algorithm chosen and perform constrained optimization. Hence, a novel generic framework has been developed which is designed for use with any agent-based optimizer and is presented in Sect. 4 of this paper. This new framework is coupled to four different agent-based optimization algorithms, and compared to using other commonly employed constraint handling frameworks for purely analytical problems (Sect. 5) and some engineering benchmark problems (Sect. 6). It is also demonstrated on a constrained aerodynamic shape optimization problem (Sect. 7). The following two

sections first present a brief outline of agent-based optimization and current methods for handling constraints.

2 Agent-based search algorithms

This section introduces the agent-based search algorithm system. A system of agents that form an agent-based search algorithm is made up of N individuals. The nth agent in the population has a location within the search space defined by a vector of design variables $\mathbf{x}_n = [x_n^1, x_n^2, \ldots, x_n^D]^T$ where D is the number of design variables), which has an initial position within the search space bounds hence $\mathbf{L} \leq \mathbf{x}_n(0) \leq \mathbf{U}$. The optimizers used in this work are briefly outlined below, with references provided for more in-depth discussions of the algorithms. It is important to note that the motivation of the work presented herein is not whether one swarm algorithm is superior to another, but it is the development and performance analysis of the new framework when coupled to a number of different optimizers. A comprehensive review and archive of swarm intelligence can be found in Engelbrecht (2005).

2.1 Particle swarm optimization (PSO)

The first algorithm considered is PSO (Kennedy and Eberhart 1995). PSO uses knowledge of the cognitive (individual) and social (population) history of the search to construct a search procedure. Equation 2 gives the velocity expression used in a basic PSO optimizer which is used to calculate the movement of an agent.

$$\mathbf{v}_n(t+1) = w(t)\mathbf{v}_n(t) + c_1\mathbf{r}_{1_n}(\mathbf{p}_n - \mathbf{x}_n(t)) + c_2\mathbf{r}_{2_n}(\mathbf{s} - \mathbf{x}_n(t)) \tag{2}$$

In Eq. 2 the subscript n is the variable for the nth agent in a swarm of N agents; w is the inertia weight; $\mathbf{p}_n$ is best position found by the nth agent so far during the optimization, which gives the cognitive aspect of the search; $\mathbf{s}$ is the best position found by the whole swarm so far (it should be noted that this location is often denoted as $\mathbf{g}$, but to avoid using the same nomenclature to represent constraints, the symbol $\mathbf{s}$ is used throughout this paper) which gives the social aspect of the search. The individuals' and swarm's best positions give the algorithm a memory quality that help drive it towards the globally optimal solution. The vectors, $\mathbf{r}_{1_n}$ and $\mathbf{r}_{2_n}$ are made up of D different random numbers (i.e., for each dimension, a new random number is used) that add a stochastic nature to the algorithm and are uniformly distributed between 0 and 1. The constants, c_1 and c_2, are the cognitive and social parameters respectively which give the local and global search extent of the scheme. Full reviews of implementations and performance aspects of PSO are presented in Poli et al. (2007) and Clerc (2013).

2.2 Gravitational search algorithm (GSA)

The second algorithm considered is the gravitational search algorithm (GSA) (Rashedi et al. 2009). GSA is an agent-based global search algorithm for

unconstrained global optimization where the principles of basic Newtonian mechanics act as the basis on which the algorithm is constructed. The agents in GSA act as masses, where an agent's mass is related to its fitness. This information is propagated through the population by global gravitational attractive forces which act as a vehicle to allow each agent in the population to have knowledge of the fitness of all other agents in the population, leading to an efficient optimization process.

The force acting on the agents is controlled by a gravitational constant, which exponentially decays from $G(0)$ at the start of the optimization procedure by a decay constant α. An acceleration is then calculated from the force acting on an agent and its mass, from which the movement can be calculated.

GSA has been considered in this work to test constraint handling on a swarm optimizer that is constructed slightly differently to PSO; in GSA there is a global transfer of data between all agents in the population. The number of computations to evaluate the movement of agents is slightly higher than PSO, however, it has been shown to be an effective algorithm.

2.3 Hybrid gravitational search particle swarm (HGSAPSO)

Individually, both PSO and GSA are effective at performing global optimization, however, researchers have also considered the performance of a hybrid version of PSO and GSA (Mirjalili et al. 2012; Tsai et al. 2013). This hybrid GSA/PSO algorithm has no standard name in the literature so for clarity, in this paper it is referred to as hybrid-GSA-PSO (HGSAPSO) and is the third algorithm considered. The HGSAPSO algorithm merges the memory qualities of PSO with the global knowledge qualities of GSA to provide a search algorithm that is superior to both.

The two are merged by adding together a weighted combination of the acceleration from PSO, $\mathbf{a}_{pso_n}$ (which is the right two terms of the velocity of PSO), which is given by:

$$\mathbf{a}_{pso_n}(t) = c_1\mathbf{r}_{1_n}(\mathbf{p}_n - \mathbf{x}_n(t)) + c_2\mathbf{r}_{2_n}(\mathbf{s} - \mathbf{x}_n(t)) \tag{3}$$

with the acceleration from GSA, $\mathbf{a}_{gsa_n}$, to give:

$$\mathbf{a}_n(t) = W\mathbf{a}_{gsa_n}(t) + (1 - W)\mathbf{a}_{pso_n}(t) \tag{4}$$

where W provides tuning between the memory qualities of PSO (which are emphasised more for lower W) and the knowledge transfer qualities of GSA (which are emphasised more for higher W). The velocity and movement are calculated using the same method as GSA, hence, if $W = 0$ then the PSO velocity equation can be recovered with a random inertia term.

2.4 Differential evolution

Differential evolution (DE) (Storn and Price 1995) is the final algorithm considered in this work. It is a swarm intelligence algorithm built around the concept of evolutionary mechanics and is specifically designed for continuous optimization. An

in-depth review of the use of DE and its developments is given in Das and Suganthan (2011). To summarise, DE uses mutation, crossover and selection steps to create candidate solutions, child solutions and new generations. The mutation stage involves the production of a new, candidate solution to introduce variability and exploration into the algorithm. The new solution is a weighted combination of three randomly chosen agents within the swarm, where a factor F is used to control the mutation. The crossover stage is used to enhance diversity in the population by combining aspects of the new and old solutions based on a crossover probability, CR. The new solution is accepted if its fitness is better than the old solution.

3 Constraint handling in agent-based search algorithms

Constraint handling is the name of the group of approaches developed to solve constrained optimization problems using agent-based optimization algorithms. The most common method are by penalty functions, special operators or separation approaches. These three methods are briefly outlined here, though a comprehensive review of constraint handling in nature-inspired numerical optimization algorithms has been presented previously (Mezura-Montes and Coello Coello 2011).

The principle of penalty functions is that a constrained optimization problem can be transformed into an unconstrained one by incorporating the constraints in an augmented objective function. The augmented objective function can then be directly solved by the optimizer. Common approaches are a death-penalty (Hu and Eberhart 2002), static penalty (Venter and Sobieszczanski-Sobieski 2003) or dynamic penalty (Parsopoulos and Vrahatis 2002). Penalty functions tend to be simple to implement and therefore are a good option for users wishing to obtain an optimized result quickly, however, the performance requires careful selection of the penalty parameters. Most penalty approaches are generally applicable to any agent-based search algorithm and this makes their performance particularly important for comparison in this work.

The special operators work on the principle that a feasible solution is better than an infeasible one, and as such manipulate the underlying search algorithm to drive the solution towards the feasible space. The first methods of this nature were based on the idea of feasible directions (Vanderplaats 1999), and have since been developed into more sophisticated approaches (Venter and Sobieszczanski-Sobieski 2003; Sun et al. 2011; Lu and Chen 2008). An obvious prerequisite of these operators is the requirement for at least one agent to be initially feasible. This can be problematic when the feasible space is small compared to the whole design space, or if equality constraints are present, where the feasible design space is a $D-1$ manifold, i.e., a line for two dimensional space. Furthermore, they also tend to alter the natural self-organising swarm dynamics that cause the algorithm to be effective at optimizing search spaces.

Finally, separation approaches are also common. These are the antithesis to penalty functions, where instead of combining the constraint violation and objective function, the two are optimized separately (Lu and Chen 2006; Liang and Suganthan 2006). These methods can generally be split into “hard feasibility rules” (binary

tournament selection is an example of this Deb 2000) and "soft feasibility rules" (α-constrained (Takahama and Sakai 2005a) and ϵ-constrained (Takahama and Sakai 2005b) methods are examples). Care has to be taken with methods such as these that premature convergence is not obtained, or that many user-defined tuning parameters are not added.

Many constraint handling techniques have been proposed for coupling to agent-based search algorithms, however, not all techniques are suitable for coupling to all agent-based optimizers. GSA, for example, is particularly difficult to couple to many constraint handling techniques. The global transfer of data that occurs in GSA (due to the global attractive forces) means data can be transferred from the infeasible to the feasible region. In reality, what this means is that if, say a penalty function is used, then the masses of the feasible agents may only vary by a small amount due to the (possibly) very large difference in objective function that can exist between the feasible and infeasible agents. This can severely restrict the ability for GSA to locate the feasible globally optimal solution. Furthermore, GSA is an example of a swarm algorithm that uses the fitness function directly in its update scheme. This makes GSA (and its derivatives such as HGSAPSO) difficult to couple to both traditional and state-of-the-art constraint handling frameworks without compromising the performance of either GSA or the constraint handling framework.

GSA is used here as an example but this argument can be made for other agent-based algorithms too, hence it would be advantageous for a constraint-handling framework to be compatible with any agent-based algorithm. A user of the framework could then wrap it around any swarm algorithm chosen and perform constrained optimization. This is the point that is dealt with in this paper and has led to the development of a generic constraint handling framework that is suitable for coupling to any agent-based search algorithm.

4 Proposed constraint handling framework

The development of the framework proposed in this work is driven by the requirement to have a general constraint handling method that can be used for any agent-based optimization algorithm. Ideally, modifications to the fundamentals of a given optimizer should be avoided as this can alter the underlying self-organising swarm intelligence of the algorithm, but instead, the constraint handling framework should be able to be added on to an existing optimizer. The approach that is presented here adheres to this requirement and works with the general agent-based optimizer system. Discussion points that are considered in the framework's development process are laid out below, followed by a formal explanation of the constraint handling framework itself.

4.1 Development points for new framework

The requirement highlighted in this paper is to have a generic constraint handling framework to be used with any agent-based optimizer, i.e., a user should be able to add this framework onto an already existing unconstrained optimizer. Therefore, to

be able to do this it is proposed that the total population of N agents be split into two independent sub-swarms, where one sub-swarm contains entirely feasible agents and the other is entirely infeasible agents. The constraint violation, $\phi(\mathbf{x}_n)$ is defined as follows:

$$\phi(\mathbf{x}_n) = \sum_{k=1}^{G} \langle g_k(\mathbf{x}_n) \rangle + \sum_{k=1}^{H} |h_k(\mathbf{x}_n)| \tag{5}$$

where G and H are the number of inequality and equality constraints respectively, and $\langle g_k(\mathbf{x}_n) \rangle$ is the constraint violation of the inequality constraint. An infeasible agent is defined as being one where $\phi(\mathbf{x}_n) > 0$. At any iteration during the search there will be N_f feasible and $N_\times$ infeasible agents (Eq. 6 must therefore hold). It should be noted, however, that N_f and $N_\times$ may change at every iteration as agents migrate from the feasible to infeasible space and *vice-versa*.

$$N = N_f + N_\times \tag{6}$$

By splitting the population into sub-swarms, the constrained optimization problem can also be split into two separate problems. Hence, the framework uses the problem separation approach where the feasible agents optimize the value of the objective function and the infeasible agents optimize the constraint violation, as shown in Eq. 7 where ζ represents the objective function of an agent; this formulation necessitates the use of sub-swarms.

$$\zeta(\mathbf{x}_n) = \begin{cases} f(\mathbf{x}_n) & \text{if} \phi(\mathbf{x}_\mathrm{n}) \leq 0 \\ \phi(\mathbf{x}_n) & \text{else} \end{cases} \tag{7}$$

The solution that the infeasible swarm is searching for is located at every point in the feasible region, hence this acts as a mechanism to firstly find the feasible space, and secondly keep the agents there. The solution that the feasible swarm is searching for is at the global minimum of the feasible region, which is the solution to the constrained optimization problem given by statement 1, assuming a feasible solution exists.

By considering sub-swarms, separate agent-based search algorithms can be used for each of the two sub-problems. Hence, a user's already existing unconstrained optimizer can be used to solve the feasible part of Eq. 7 while being independent of the infeasible part. This avoids unwanted dilution of the feasible data by the infeasible data and promotes the performance qualities of a user's swarm optimizer. To solve Eq. 7 for the infeasible sub-swarm any agent-based search algorithm can be used, however, for this constraint handling framework, a simple PSO is suggested to take advantage of the memory components. The $\mathbf{p}_n$ and $\mathbf{s}$ positions, which are the individual and global best positions found so far should be made feasible if they can be feasible to allow a velocity component to point towards the feasible region. This can be done using a binary tournament selection procedure. While a simple PSO movement procedure is suggested, in fact, a user could use any swarm algorithm for the unconstrained problem too. This would add further flexibility to the framework.

4.2 Separation-sub-swarm (3S) framework

The constraint handling framework described in this paper is hereafter called separation-sub-swarm (3S), and is fully outlined in this section for completion. At any iteration, t, during the optimization there are a constant number of N agents in the whole swarm, where an individual agent is the nth agent of the whole swarm. The objective function and constraint values of each agent must be calculated, and from that the population can be split into a sub-swarm containing N_f feasible agents (all constraints are satisfied) and $N_\times$ infeasible agents (at least one constraint is violated). An individual agent in the feasible sub-swarm is the ith feasible agent, whereas an individual in the infeasible sub-swarm is the jth infeasible agent.

The objective function of the ith agent in the feasible swarm is:

$$\zeta(\mathbf{x}_i) = f(\mathbf{x}_i) \tag{8}$$

and the objective function of the jth agent in the infeasible swarm is:

$$\zeta(\mathbf{x}_j) = \phi(\mathbf{x}_j) \tag{9}$$

The movement procedure of the infeasible agents in the population is by a simple PSO so all agents in the population (including the feasible ones) must have their best position, $\mathbf{p}_n$, updated. The swarm's best position, $\mathbf{s}$, also needs to be updated. A domination procedure based on a tournament selection is used, where the domination operator $\prec$ is used to determine the domination between two locations in the search space, $\mathbf{x}_a$ and $\mathbf{x}_b$. The domination operator then works as follows:

$$\mathbf{x}_a \prec \mathbf{x}_b \Leftrightarrow \begin{cases} f(\mathbf{x}_b) < f(\mathbf{x}_a) \text{ and } \phi(\mathbf{x}_\mathrm{a}), \phi(\mathbf{x}_\mathrm{b}) \leq 0 \\ \phi(\mathbf{x}_b) \leq 0 \text{ and } \phi(\mathbf{x}_\mathrm{a}) > 0 \\ \phi(\mathbf{x}_b) < \phi(\mathbf{x}_a) \text{ and } \phi(\mathbf{x}_\mathrm{a}), \phi(\mathbf{x}_\mathrm{b}) > 0 \end{cases} \tag{10}$$

and

$$\mathbf{x}_b \mapsto \mathbf{x}_a \Leftrightarrow \mathbf{x}_a \prec \mathbf{x}_b \tag{11}$$

Hence, if $\mathbf{x}_a$ is dominated by $\mathbf{x}_b$ then $\mathbf{x}_a$ is updated with $\mathbf{x}_b$. $\mathbf{p}_n \prec \mathbf{x}_n$ and $\mathbf{s} \prec \mathbf{x}_n$ are used to update $\mathbf{p}_n$ and $\mathbf{s}$. The velocity and movement of the jth infeasible agent is then calculated as:

$$\mathbf{v}_j(t+1) = \mathbf{r}_{0_j}\mathbf{v}_j(t) + c_1\mathbf{r}_{1_j}(\mathbf{p}_j - \mathbf{x}_j(t)) + c_2\mathbf{r}_{2_j}(\mathbf{s} - \mathbf{x}_j(t)) \tag{12}$$

$$\mathbf{x}_j(t+1) = \mathbf{x}_j(t) + \mathbf{v}_j(t) \tag{13}$$

where $\mathbf{r}_{0_j}$ is a D long vector of random numbers, analogous to $\mathbf{r}_{1_j}$ and $\mathbf{r}_{2_j}$. The pseudocode of the 3S framework is given in Algorithm 1 where the highlighted line of the code is where the user would add their selected agent-based search algorithm for updating the feasible swarm. This demonstrates that the 3S framework is compatible with any agent-based optimizer with no modification required to the mechanics of a user's swarm algorithm.

Algorithm 1 3S framework

```
Initialise agents with random positions and velocities
for t = 1 → T do
    Calculate f(x), g(x) and h(x) of all agents
    Calculate φ(x) and ζ(x) of all agents
    Separate feasible and infeasible agents
    for n = 1 → N do
        x_n ↦ p_n ⇔ p_n ≺ x_n
        x_n ↦ s ⇔ s ≺ x_n
    end for
    for j = 1 → N_× do
        Update v_j of all infeasible agents: equation 12
        Update x of all infeasible agents: equation 13
    end for
    for i = 1 → N_f do
        Update x_i of all feasible agents by selected agent-based method
    end for
end for
```

When implementing the 3S framework to handle constraints with any swarm optimizer, tracking of the swarm's best position means that the user can track the feasibility of the optimum design found so far. At the end of the optimization, if no feasible solution has been found then the value stored in $\mathbf{s}$ is the closest solution to the feasible region that could be found.

5 Constrained analytical optimization

The 3S framework for handling constraints when using agent-based optimization algorithms is outlined fully in Algorithm 1. This framework is compatible with any agent-based algorithm, the user need only add the framework to their optimizer to be able to handle constraints. To demonstrate this, constrained analytical optimization is performed in this section for four different swarm algorithms: PSO, GSA, HGSAPSO and DE. The simple PSO with inertia is used to demonstrate the effect of using 3S with a PSO-based algorithm; it should be noted that many other PSO algorithms are also available though the goal of this work is not to prove whether one agent-based algorithm is superior to another, but to prove that the newly designed 3S framework is suitable for use with any agent-based algorithm.

5.1 Methods used for comparison

To compare the performance of 3S, other constraint handling frameworks that are also suitable for use with any swarm optimizer are also tested. These methods are a death penalty method, static penalty method, dynamic penalty method and feasible directions method. The formulations of these methods are also detailed below.

The death penalty, which is the simplest method implemented, randomly reinitialises an infeasible agent.

For the static penalty, the formulation of the augmented objective function is given as:

$$\zeta(\mathbf{x}_n) = f(\mathbf{x}_n) + \theta\phi(\mathbf{x}_n) \tag{14}$$

where θ is the static penalty factor. The penalty factor was chosen based on comparing three orders of magnitude factors ($\theta = 1, 10, 100$) on the G7 problem using PSO (details of runs are found later in Sect. 5.2). When $\theta = 1$, from the 25 runs, not one was found feasible and it appears that this factor is not large enough to create a penalty suitable to find a feasible solution. However, when $\theta = 10$ or $\theta = 100$ the feasibility rate was 100%, but the standard deviation of the $\theta = 10$ factor was approximately half of when $\theta = 100$ indicating that a too high penalty factor weights feasibility over exploitation and therefore inhibits a good optimum being found. Hence, a penalty factor of $\theta = 10$ was chosen to give a reasonable balance between feasibility and exploitation.

The more complicated dynamic penalty formulation (Parsopoulos and Vrahatis 2002) is:

$$\zeta(\mathbf{x}_n) = f(\mathbf{x}_n) + \kappa\left(\sum_{k=1}^{G}\theta_k\langle g_k(\mathbf{x}_n)\rangle^{\gamma_k} + \sum_{k=1}^{H}\theta_k|h_k(\mathbf{x}_n)|^{\gamma_k}\right) \tag{15}$$

where κ is the dynamic penalty which is given by $t\sqrt{t}$ at the tth iteration; $\theta_k = 10$ if $\langle g_k\rangle, |h_k| < 0.001$, else $\theta_k = 20$ if $\langle g_k\rangle, |h_k| < 0.1$, else $\theta_k = 100$ if $\langle g_k\rangle, |h_k| < 1.0$, otherwise $\theta_k = 300$; $\gamma_k = 1.0$ if $\langle g_k\rangle, |h_k| < 1$, otherwise $\gamma_k = 2$.

The feasible direction approach used here is based on an approach developed for PSO (Venter and Sobieszczanski-Sobieski 2003) where a manipulation of the velocity vector is made if an agent is infeasible to attempt to force it towards the feasible region. Hence for the jth infeasible particle, the velocity vector is constructed as:

$$\mathbf{v}_j(t+1) = \mathbf{r}_{0_j}(\mathbf{s}(t) - \mathbf{x}_j(t)) \tag{16}$$

where $\mathbf{s}$ is the swarm's best position found, which is feasible if possible. If this is feasible then the velocity vector will always point towards the feasible region. As

Table 1 Parameter values used in GSA, PSO, HGSAPSO and DE

Parameter	GSA value	PSO value	HGSAPSO value	DE value
N	200	200	200	50
T	1500	1500	1500	–
nf_{max}	300,000	300,000	300,000	300,000
$G(0)$	30	–	30	–
α	10	–	10	–
c_1, c_2	–	2	2	–
w	–	Rand (0, 1)	Rand (0, 1)	–
W	–	–	0.5	–
CR	–	–	–	0.3
F	–	–	–	0.7

before, the update of the swarm's best position is by the domination operator, hence $\mathbf{x}_n \mapsto \mathbf{s} \Leftrightarrow \mathbf{s} \prec \mathbf{x}_n$.

5.2 Run details

The parameter values used for GSA, PSO, HGSAPSO and DE are given in Table 1 where N is the number of agents, T is maximum number of iterations, $G(0)$ is the initial gravitational constant, α is the gravitational decay constant, c_1 is the cognitive parameter, c_2 is the social parameter, w is the inertia factor, and W controls the influence of GSA and PSO in HGSAPSO respectively. These values were chosen as they reflected parameter values found in the literature and gave all constraint handling frameworks good performance. If any agent exits the bounds of the design space, i.e., if $\mathbf{x}_i \notin \mathbf{S}$ (as given in statement 1) then it is reinitialised in its last position. The stopping criteria is until the number of function evaluations reaches nf_{max}. This is kept constant through all the tests to ensure fairness of comparison between the different constraint handling techniques.

The CEC2006 constrained analytical function suite (Liang et al. 2013) is used in this paper for testing the performance of the constraint handling techniques. The suite contains 24 test cases that are all minimization problems and contain various numbers of linear and non-linear inequality and equality constraints, various sizes of feasible search space, and various types of objective function. Cases 20 and 22 are omitted due to not having a feasible solution within the bounds of the stated design space. The nature of the cases are outlined in Table 2. Equality constraints are transformed into inequality constraints to within a small tolerance: $|h_j(\mathbf{x})| - \epsilon \leq 0$.

5.3 Results

25 independent runs of each of the test functions using each constraint handling framework with each swarm algorithm were performed when testing. The degree of violation for the equality constraints was $\epsilon = 0.0001$, so as a result optimum solutions better than the theoretical optimum were possible. The error was calculated as the difference between the best feasible solution and the analytical minimum objective value stated in the CEC 2006 definitions (Liang et al. 2013). The final results were ranked based on the final error in the solution. Any infeasible solutions were ranked after any feasible ones, and by their final constraint violation. The best result (the one with the lowest error) of the 25 runs using the five different constraint handling methods with GSA, PSO, HGSAPSO and DE are presented in Tables 3, 4, 5 and 6. Also presented are the standard deviation of all of the feasible results and the number of runs that resulted in a feasible solution. If an infeasible solution results from any of the runs then the worst value is presented as INF, meaning infeasible. If feasible solutions are available then the best results presented are from the feasible solutions only, hence if no feasible solution can be found from the 25 runs, then the best solutions will be infeasible. The average feasibility rate of each constraint handling technique is shown graphically in Fig. 1. The feasibility rate is the percentage of runs that found a feasible solution. Example convergence

Table 2 Summary of 24 analytical test cases, where ρ is the ratio of the feasible search space to the whole search space, and G_L, G_N, H_L, H_N represent the number of linear inequalities, nonlinear inequalities, linear equalities and nonlinear equalities respectively, a is number of active constraints at solution

Function	D	Type of f	ρ (%)	G_L	G_N	H_L	H_N	a	Optimal f
G1	13	Quadratic	0.0111	9	0	0	0	6	−15.0000
G2	20	Nonlinear	99.997	0	2	0	0	1	−0.8036
G3	10	Polynomial	0.0000	0	0	0	1	1	−1.0005
G4	5	Quadratic	52.123	0	6	0	0	2	−30,665.5387
G5	4	Cubic	0.0000	9	0	0	3	3	5126.4967
G6	2	Cubic	0.0066	9	2	0	0	2	−6961.8139
G7	10	Quadratic	0.0003	3	5	0	0	6	24.3062
G8	2	Nonlinear	0.8560	0	2	0	0	0	−0.0958
G9	7	Polynomial	0.5121	0	4	0	0	2	680.6301
G10	8	LInear	0.0010	3	3	0	0	6	7049.2480
G11	2	Quadratic	0.0000	0	0	0	1	1	0.7499
G12	3	Quadratic	4.7713	0	1	0	0	0	−1.0000
G13	5	Nonlinear	0.0000	0	0	0	3	3	0.0539
G14	10	Nonlinear	0.0000	0	0	3	0	3	−47.7649
G15	3	Quadratic	0.0000	0	0	1	1	2	961.7150
G16	5	Nonlinear	0.0204	4	34	0	0	4	−1.9052
G17	6	Nonlinear	0.0000	0	0	0	4	4	8853.5397
G18	9	Quadratic	0.0000	0	13	0	0	6	−0.8660
G19	15	Nonlinear	33.476	0	5	0	0	0	32.6556
G20	24	Linear	0.0000	0	6	2	12	16	Infeasible
G21	7	Linear	0.0000	0	1	0	5	6	193.7245
G22	22	Linear	0.0000	0	1	8	11	19	Infeasible
G23	9	Linear	0.0000	0	2	3	1	6	−400.0551
G24	2	Linear	79.656	0	2	0	0	2	−5.5080

plots for the best solution found for a number of functions using different swarm algorithms are shown in Fig. 2.

The performance of the 3S constraint handling framework compared to the other generic frameworks of penalty functions and feasible directions demonstrates that, overall, this new framework produces superior performance, both in terms of finding a feasible solution and finding the best feasible solution. In terms of finding a feasible solution through the optimizations, Fig. 1 demonstrates that the 3S framework, overall, outperforms all of the other methods tested. The feasibility rate for 3S is consistently above 90% for all of the swarm algorithms tested demonstrating that the 3S framework can be described as a generic constraint handling framework. The primary reason for the framework to be able to find a feasible solution at such a high rate is possibly due to the domination procedure that applies a lexicographic ordering that enforces feasibility to precede anything else. This introduces a velocity component that points, at least, towards a less infeasible

Table 3 Feasible results of 24 analytical function test suite optimized using various constraint handling methods with GSA swarm (INF means no feasible solution could be found)

	3S	Static	Death	Dyn	FD
Best					
G1	−14.9907	−14.9833	−5.5840	−14.9061	INF
G2	−0.4522	−0.4272	−0.4587	−0.4354	−0.4369
G3	−0.9920	INF	−0.0894	−0.9747	INF
G4	−30665.5354	INF	−30394.9990	−30661.8813	−30033.6496
G5	5126.5114	5313.8254	INF	INF	INF
G6	−6961.8131	INF	−6876.4704	−6717.7858	−6823.0618
G7	24.8426	26.1253	344.6819	80.9496	INF
G8	−0.0958	INF	−0.0958	−0.0958	−0.0947
G9	680.8516	683.8443	703.3757	691.9333	714.4896
G10	7184.0823	7064.7437	13794.2317	7451.3936	INF
G11	0.7499	0.7500	0.7499	0.7500	0.9217
G12	−1.0000	−1.0000	−1.0000	−1.0000	−1.0000
G13	0.1777	0.4398	INF	0.4403	INF
G14	−46.4714	INF	INF	INF	INF
G15	961.7151	963.0040	INF	961.8996	INF
G16	−1.9051	−1.3273	−1.7980	−1.3508	INF
G17	8870.2852	9782.8587	INF	INF	INF
G18	−0.8645	−0.6668	INF	−0.6467	INF
G19	34.5653	96.3559	683.0107	480.7436	327.6714
G21	672.7320	475.3641	INF	9132.8898	INF
G23	−70.0588	INF	INF	INF	INF
G24	−5.5080	−5.4960	−5.5042	−5.4904	−5.4634
Standard deviation (N. feasible runs)					
G1	0.0151 (25)	0.0327 (25)	1.7482 (14)	0.9817 (25)	−(0)
G2	0.0079 (25)	0.0288 (25)	0.0191 (25)	0.0262 (25)	0.0322 (25)
G3	0.0051 (25)	−(0)	0.0252 (13)	0.0265 (25)	−(0)
G4	0.0698 (25)	−(0)	112.8312 (25)	74.8931 (6)	432.1321 (16)
G5	7.7903 (25)	472.9426 (25)	−(0)	−(0)	−(0)
G6	0.0014 (25)	−(0)	450.9520 (25)	351.4827 (25)	561.9018 (24)
G7	0.1608 (25)	0.5597 (25)	987.8801 (7)	57.3000 (25)	−(0)
G8	0.0000 (25)	−(0)	0.0000 (25)	0.0000 (25)	0.0273 (24)
G9	0.1220 (25)	1.3350 (25)	21.7667 (25)	10.6778 (25)	167.2007 (25)
G10	58.3379 (25)	303.2353 (3)	1022.9673 (8)	1281.6319 (24)	−(0)
G11	0.0000 (25)	0.0020 (25)	0.0021 (25)	0.0269 (25)	0.0000 (1)
G12	0.0000 (25)	0.0000 (25)	0.0000 (25)	0.0000 (25)	0.0000 (25)
G13	0.1516 (25)	0.1790 (25)	−(0)	0.1874 (25)	−(0)
G14	0.8196 (25)	−(0)	−(0)	−(0)	−(0)
G15	0.2823 (25)	0.7276 (25)	−(0)	1.6090 (25)	−(0)
G16	0.0000 (25)	14.2727 (25)	0.0775 (25)	2905.0956 (18)	−(0)
G17	21.9869 (25)	511.1305 (25)	−(0)	−(0)	−(0)

Table 3 continued

	3S	Static	Death	Dyn	FD
G18	0.0018 (25)	0.0096 (25)	−(0)	0.0709 (25)	−(−(0)
G19	0.4775 (25)	17.4433 (25)	73.9192 (25)	51.3574 (25)	493.9967 (25)
G21	0.0000 (1)	27.8311 (25)	−(0)	0.0000 (1)	−(0)
G23	161.0525 (7)	−(0)	−(0)	−(0)	−(0)
G24	0.0000 (25)	0.0102 (25)	0.0146 (25)	0.0182 (25)	0.1161 (25)

solution, or towards feasibility is a feasible solution is known. Secondary to this is also the separation that takes place in the algorithm which inhibits data being transferred from the infeasible search to the feasible search. This benefits the search properties of the user specified swarm algorithm, which are promoted.

A clear picture of the quality and repeatability of the final results obtained can be sought by considering the standard deviation and the number of feasible runs. In particular, the 3S framework has consistently small standard deviations and large numbers of results that gave a feasible solution and is homogeneous across the four swarm algorithms tested. It is interesting to note, that when coupling to DE, 3S provides best solutions that are, overall, closer to the best available solution than the other swarm algorithms. Another example of where DE has high performance is when using feasible direction with DE. In general, the feasible directions framework has poor results both in terms of feasibility, and quality of the final solution. This performance is, however, vastly improved when considering DE as the swarm algorithm, over PSO, GSA and HGSAPSO.

Comparing Fig. 2a with b and e with f allows comparisons of the performance of the constraint handling frameworks on problems containing different numbers of inequality and equality constraints. Figure 2a is the convergence of the frameworks with GSA on problem G11, which has 1 equality constraint, whereas, Fig. 2b is the convergence of the frameworks with GSA on problem G16, which has 38 inequality constraints. An equality constrained problem effectively acts as an optimization along a line, hence represents a difficult optimization problem. The 3S framework, however, has performed well in both cases, producing a final error that is multiple orders of magnitude lower than the other frameworks for both cases. This is a trend that is seen when considering the use of DE as the swarm algorithm too. Comparing Fig. 2e (1 equality constraint) with Fig. 2f (2 inequality constraints) shows, again, that the 3S framework performs well with both inequality and equality constrained problems.

If rate of convergence is a limiting factor for making a choice of which constraint handling framework to use then the 3S framework allows the limiting factor of the convergence to be driven by the attached swarm optimizer. This is particularly useful in examples where the number of objective function evaluations should be kept to a minimum. This behaviour is demonstrated by considering the different rates of convergence of Fig. 2c (HGSAPSO), d (PSO) and f (DE). The convergence rates of all three of these optimizers are considerably different, and are driven by the

Table 4 Feasible results of 24 analytical function test suite optimized using various constraint handling methods with PSO swarm (INF means no feasible solution could be found)

	3S	Static	Death	Dyn	FD
Best					
G1	−15.0000	−15.0000	−5.7498	−15.0000	INF
G2	−0.8036	−0.8036	−0.7977	−0.8035	−0.7460
G3	−0.9943	INF	−0.0570	−0.9683	INF
G4	−30665.5395	INF	−30660.7782	−30663.8131	−30604.0093
G5	5126.4985	5126.5266	INF	5126.5551	INF
G6	−6961.8139	INF	−6825.0976	−6961.8139	INF
G7	24.3452	24.3412	489.5621	24.3400	INF
G8	−0.0958	INF	−0.0958	−0.0958	−0.0958
G9	680.6308	680.6313	683.7978	680.6337	INF
G10	7051.8133	INF	12858.4867	7151.1207	INF
G11	0.7499	0.7499	0.7500	0.7499	INF
G12	−1.0000	−1.0000	−1.0000	−1.0000	−1.0000
G13	0.0928	0.2399	INF	0.7805	INF
G14	−45.9470	INF	INF	INF	INF
G15	961.7168	961.7165	INF	961.7349	INF
G16	−1.9052	−1.9052	−1.8072	−1.9052	INF
G17	8866.0629	8873.2060	INF	8874.6330	INF
G18	−0.8660	−0.8660	INF	−0.8660	INF
G19	32.9288	33.7257	92.8825	33.4726	52.0220
G21	INF	208.6892	INF	3134.5820	INF
G23	−305.4569	INF	INF	4728.5811	INF
G24	−5.5080	−5.5080	−5.5077	−5.5080	−5.5080
Standard deviation (N. feasible runs)					
G1	0.0000 (25)	0.8956 (25)	1.6716 (15)	0.8956 (25)	−(0)
G2	0.0041 (25)	0.0220 (25)	0.0446 (25)	0.0370 (25)	0.0525 (25)
G3	0.0100 (25)	−(0)	0.0173 (13)	0.1290 (25)	−(0)
G4	0.0000 (25)	−(0)	10.9913 (25)	10.8116 (3)	169.9077 (9)
G5	5.8372 (25)	33.4327 (25)	−(0)	290.2647 (21)	−(0)
G6	0.0000 (25)	−(0)	455.2729 (25)	0.0005 (14)	−(0)
G7	0.0554 (25)	0.2703 (25)	928.9641 (10)	0.5058 (25)	−(0)
G8	0.0000 (25)	−(0)	0.0000 (25)	0.0000 (25)	0.0000 (3)
G9	0.0013 (25)	0.0075 (25)	6.4125 (25)	0.0083 (25)	−(0)
G10	72.4586 (25)	−(0)	1519.0320 (7)	795.2348 (16)	−(0)
G11	0.0000 (25)	0.0000 (25)	0.0055 (25)	0.0000 (25)	−(0)
G12	0.0000 (25)	0.0000 (25)	0.0000 (25)	0.0000 (25)	0.0000 (25)
G13	0.2274 (25)	0.2443 (25)	−(0)	223.8461 (25)	−(0)
G14	0.5747 (25)	−(0)	−(0)	−(0)	−(0)
G15	0.3319 (25)	0.4739 (25)	−(0)	3.3241 (25)	−(0)
G16	0.0000 (25)	0.0000 (25)	0.0617 (25)	0.0000 (25)	−(0)
G17	24.2533 (25)	76.6460 (25)	−(0)	741.1933 (16)	−(0)

Table 4 continued

	3S	Static	Death	Dyn	FD
G18	0.0008 (25)	0.0923 (25)	−(0)	0.0888 (18)	−(0)
G19	0.4927 (25)	1.7673 (25)	55.9256 (25)	1.7651 (25)	81.0963 (12)
G21	−(0)	82.0738 (13)	−(0)	2162.5011 (4)	−(0)
G23	174.9549 (11)	−(0)	−(0)	1428.6489 (3)	−(0)
G24	0.0000 (25)	0.0000 (25)	0.0006 (25)	0.0000 (25)	0.4636 (13)

chosen swarm algorithm. These three figures also demonstrate, unlike the other methods, that the 3S framework does not restrict the high rates of convergence seen with methods such as DE and PSO. This functionality of the 3S framework comes about due to splitting the population into independent swarms depending on feasibility. The coupling of other high performance swarm optimizers is therefore facilitated without restricting the performance of these optimizers.

6 Engineering benchmark problems

The 3S framework is also tested on a number of common mechanical engineering benchmark problems. Again, as with the CEC2006 suite of problems, the 3S framework is used with four different swarm algorithms (GSA, PSO, HGSAPSO and DE) and is compared to the other generic constraint handling methods outlined in this paper (death penalty, static penalty, dynamic penalty and feasible directions). The three engineering benchmark cases investigated represent commonly used problems to investigate algorithm performance. The chosen problems are (1) welded beam design (Rao 2013) (2) pressure vessel design (Kannan and Kramer 1994) and (3) spring design (Belegundu and Arora 1985).

A summary of the number of design variables, design variable designations and number of constraints (all are inequalities) are shown in Table 7. The problem geometries are shown in Figs. 3, 4 and 5 respectively. Hence, the welded beam problem has four continuous design variables which are weld thickness h, weld length l, bar thickness t and bar width b; the pressure vessel problem has design variables are the thickness of the cylinder, T_s, the thickness of the head, T_h, the radius of the cylinder, R, and the length of the cylinder L; the spring problem variables are the number of coils of the spring, N (i.e., the total length of the material to produce the spring), the maximum diameter of the spring, D, and the thickness of the material, d. The pressure vessel problem is a mixed integer problem. That is, x_1 and x_2 must be integer multiples of 0.0625 to represent the different gauges of steel available. In the implementation of the swarm algorithms an integer design variable is derived from a continuous set by rounding the continuous variable to the nearest integer multiple of 0.0625. It should be noted that the four constraints are not related to the requirement of having integer variables, and these constraints are separate to this requirement.

Table 5 Feasible results of 24 analytical function test suite optimized using various constraint handling methods with HGSAPSO swarms (INF means no feasible solution could be found)

	3S	Static	Death	Dyn	FD
Best					
G1	−15.0000	−15.0000	−5.1698	−15.0000	INF
G2	−0.8036	−0.8036	−0.8006	−0.8036	−0.7909
G3	−0.9925	INF	−0.1834	−1.0000	INF
G4	−30665.5394	INF	−30662.5644	−30626.3635	−30381.0190
G5	5126.5107	5126.8962	INF	5131.7650	INF
G6	−6961.8139	INF	−6623.7927	−6961.8138	−6961.8060
G7	24.3128	24.3515	206.6377	24.3281	INF
G8	−0.0958	INF	−0.0958	−0.0958	−0.0958
G9	680.6302	680.6310	682.8158	680.6310	697.3279
G10	7051.7103	INF	11877.3682	7326.3568	INF
G11	0.7499	0.7499	0.7500	0.7499	INF
G12	−1.0000	−1.0000	−1.0000	−1.0000	−1.0000
G13	0.1051	0.4484	INF	11.3736	INF
G14	−45.8997	INF	INF	INF	INF
G15	961.7163	961.7151	INF	961.7253	INF
G16	−1.9052	−1.9052	−1.8046	−1.9052	INF
G17	8859.7080	8855.1521	INF	13226.0210	INF
G18	−0.8660	−0.8660	INF	−0.8660	INF
G19	33.0281	42.9999	86.0055	48.1947	67.2288
G21	INF	195.5876	INF	791.1765	INF
G23	−230.7389	INF	INF	5850.4888	INF
G24	−5.5080	−5.5080	−5.5080	−5.5080	−5.5080
Standard deviation (N. feasible runs)					
G1	0.0000 (25)	0.0000 (25)	1.4675 (16)	0.0000 (19)	−(0)
G2	0.0039 (25)	0.0318 (25)	0.0434 (25)	0.0356 (25)	0.0525 (25)
G3	0.0069 (25)	−(0)	0.0447 (18)	0.0047 (25)	−(0)
G4	0.0000 (25)	−(0)	7.5015 (25)	73.9682 (5)	313.0055 (11)
G5	7.5209 (25)	113.7252 (25)	−(0)	2575.4534 (16)	−(0)
G6	0.0007 (25)	−(0)	571.4141 (25)	0.0007 (25)	20.8651 (19)
G7	0.0148 (25)	0.1781 (25)	906.4599 (9)	0.1731 (24)	−(0)
G8	0.0000 (25)	−(0)	0.0000 (25)	0.0000 (25)	0.0000 (23)
G9	0.0004 (25)	0.0037 (25)	2.3130 (25)	0.0058 (25)	0.0000 (1)
G10	43.7134 (25)	−(0)	1459.8375 (6)	1315.2868 (15)	−(0)
G11	0.0000 (25)	0.0000 (25)	0.0018 (25)	0.0055 (25)	−(0)
G12	0.0000 (25)	0.0000 (25)	0.0000 (25)	0.0000 (25)	0.0000 (25)
G13	0.1495 (25)	0.1813 (25)	−(0)	332.9316 (25)	−(0)
G14	0.6098 (25)	−(0)	−(0)	−(0)	−(0)
G15	0.1202 (25)	0.0654 (25)	−(0)	733.8463 (25)	−(0)
G16	0.0000 (25)	0.0283 (25)	0.0650 (25)	0.0359 (25)	−(0)
G17	24.8150 (25)	40.4461 (25)	−(0)	1200.3351 (2)	−(0)

Table 5 continued

	3S	Static	Death	Dyn	FD
G18	0.0001 (25)	0.0824 (25)	−(0)	0.0835 (24)	−(0)
G19	0.1868 (25)	11.1215 (25)	42.6108 (25)	12.1735 (25)	117.1247 (24)
G21	−(0)	69.6570 (25)	−(0)	2730.0626 (8)	−(0)
G23	187.4413 (13)	−(0)	−(0)	1222.5503 (10)	−(0)
G24	0.0000 (25)	0.0000 (25)	0.0002 (25)	0.0000 (25)	0.0000 (25)

Again, the various constraint handling frameworks combined with the swarm optimizers are run on the three benchmark problems 25 times to obtain a distribution of results. The parameter values used in the algorithms are those used in the CEC2006 test (Table 1). The results of the runs for each of the cases are shown in Tables 8, 9 and 10 and Fig. 6. Tables 8, 9 and 10 also gives the number of constraints that satisfy a given tolerance, which have been taken from the best solution of the engineering benchmark cases, i.e., this is the total number of constraints that satisfy $g_k(\mathbf{x}) \leq e$ for the best solution, where e is the chosen tolerance. The discussion of the results follows in the next sections after the definition of each of the problems. To obtain a comparison of how the 3S method is working, as well as comparing to the other constraint handling methods outlined in this paper, previously published results are also given in a subsequent section.

6.1 Welded beam design

The results of the 25 runs using each of the constraint handling frameworks with each of the swarm algorithms is shown in Table 8. It can be seen, like in the CEC2006 runs, that the 3S framework performs consistently better than the other frameworks. The best, median and mean solutions for all the swarm algorithms is regularly lower when using the 3S framework. Furthermore, the standard deviation is always low, indicating that 3S is a very consistent algorithm compared to the others tested. Again, feasible directions performs the poorest of the constraint handling methods tested, however, the static penalty performs on par with 3S when using the PSO and HGSAPSO algorithms. It is also clear, from Table 8 that the 3S framework is able to converge the active constraint values right down when coupled with both DE and PSO.

An example convergence history of the welded beam problem when using DE with the five constraint handling frameworks is shown in Fig. 6a. Only the first 20000 function evaluations are shown just to demonstrate the convergence. It can be seen that when using 3S for constraint handling, the high convergence rate associated with DE is not compromised and the solution is converged onto within only a few iterations of the optimization. An optimum solution is converged onto in less than 15,000 function evaluations. A high convergence rate was also observed in the CEC2006 runs. The separation of the two swarms promotes good convergence

Table 6 Feasible results of 24 analytical function test suite optimized using various constraint handling methods with DE swarms (INF means no feasible solution could be found)

	3S	Static	Death	Dyn	FD
Best					
G1	−15.0000	−15.0000	−6.9717	−15.0000	−8.0216
G2	−0.8035	−0.8036	−0.7994	−0.8036	−0.7390
G3	−0.9799	INF	−0.4696	−0.6465	−0.0000
G4	−30665.5395	INF	−30600.4501	−30665.5395	−30432.0489
G5	5126.6494	5126.4967	INF	5126.4967	INF
G6	−6961.8139	INF	−6697.3707	−6961.8139	−5082.4156
G7	24.3107	24.3062	246.7845	24.3062	263.4170
G8	−0.0958	INF	−0.0958	−0.0958	−0.0958
G9	680.6308	680.6301	718.9932	680.6301	703.6845
G10	7051.7018	INF	14913.9644	7049.2480	INF
G11	0.7499	0.7499	0.7500	0.7499	0.7499
G12	−1.0000	−1.0000	−1.0000	−1.0000	−1.0000
G13	0.0857	0.0539	INF	0.0961	INF
G14	−45.9878	INF	INF	−47.7470	INF
G15	961.7151	961.7150	INF	961.7150	INF
G16	−1.9052	−1.9052	−1.8193	−1.9052	−1.5102
G17	8857.9984	8853.5397	INF	8854.9085	INF
G18	−0.8660	−0.8660	INF	−0.8660	INF
G19	32.9170	32.6665	34.6530	32.6587	176.7297
G21	328.9515	INF	INF	193.7245	INF
G23	−232.3515	INF	INF	−400.0551	INF
G24	−5.5080	−5.5080	−5.5080	−5.5080	−5.5079
Standard deviation (N. feasible runs)					
G1	0.0000 (25)	0.0000 (25)	1.7525 (14)	0.0000 (25)	0.5193 (24)
G2	0.0000 (25)	0.0129 (25)	0.0034 (25)	0.0148 (25)	0.0138 (25)
G3	0.0312 (25)	−(0)	0.1313 (12)	0.1438 (25)	0.0000 (1)
G4	0.0000 (25)	−(0)	42.6622 (25)	0.0000 (18)	96.1643 (25)
G5	6.8654 (25)	0.0000 (25)	−(0)	0.7296 (25)	−(0)
G6	0.0000 (25)	−(0)	412.7536 (25)	0.0000 (25)	1018.4201 (13)
G7	0.0217 (25)	0.0000 (25)	984.4026 (3)	0.0000 (25)	745.5171 (24)
G8	0.0000 (25)	−(0)	0.0000 (25)	0.0000 (25)	0.0001 (25)
G9	0.0018 (25)	0.0000 (25)	45.2053 (25)	0.0000 (25)	38.6484 (25)
G10	45.7948 (25)	−(0)	625.4232 (3)	0.0023 (23)	−(0)
G11	0.0000 (25)	0.0000 (25)	0.0029 (25)	0.0000 (25)	0.0502 (25)
G12	0.0000 (25)	0.0000 (25)	0.0000 (25)	0.0000 (25)	0.0000 (25)
G13	0.1611 (25)	0.1521 (25)	−(0)	0.1164 (25)	−(0)
G14	0.6042 (25)	−(0)	−(0)	0.4360 (25)	−(0)
G15	0.2456 (25)	0.0000 (25)	−(0)	0.0000 (25)	−(0)
G16	0.0000 (25)	0.0000 (25)	0.0750 (25)	0.0000 (25)	0.1259 (23)
G17	22.3588 (25)	0.0000 (25)	−(0)	37.3445 (25)	−(0)

Table 6 continued

	3S	Static	Death	Dyn	FD
G18	0.0005 (25)	0.0000 (25)	−(0)	0.0000 (25)	−(0)
G19	0.1466 (25)	0.0913 (25)	3.7131 (25)	0.1830 (25)	160.1440 (25)
G21	0.6786 (3)	−(0)	−(0)	38.2021 (25)	−(0)
G23	171.5676 (20)	−(0)	−(0)	8.3131 (25)	−(0)
G24	0.0000 (25)	0.0000 (25)	0.0000 (25)	0.0000 (25)	0.0106 (25)

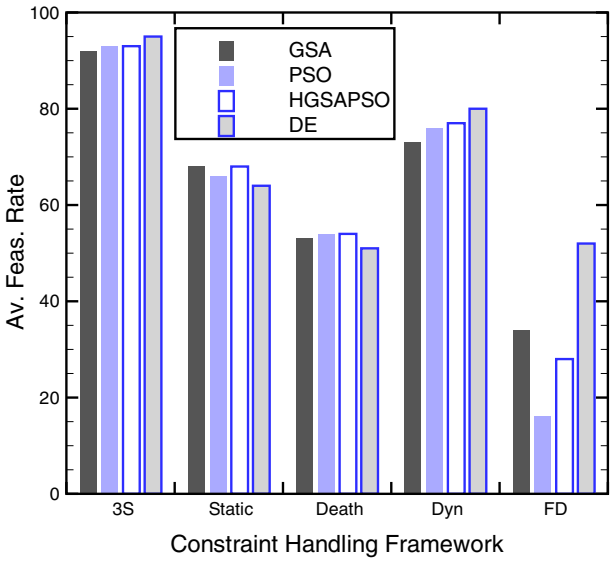


Fig. 1 Average feasibility rates

characteristics by allowing the feasible swarm to inherit the convergence characteristics of the parent swarm algorithm, and this is an advantage of using 3S.

6.2 Pressure vessel design

The results of the tests of running the optimizations using each of the swarm algorithms with each of the constraint handling frameworks 25 times are shown in Table 9. It can be seen that over all four of the swarm algorithms tested that the 3S framework has outperformed the other constraint handling frameworks. It has produced lower best solutions as well as smaller standard deviations, again indicative of an effective algorithm. The death penalty constraint handling method when used with DE is the only method over all of those tested that performs on par with the 3S framework. Convergence tolerance is similar to the welded beam examples, again showing that 3S is able to converge down the active constraints to very small tolerances.

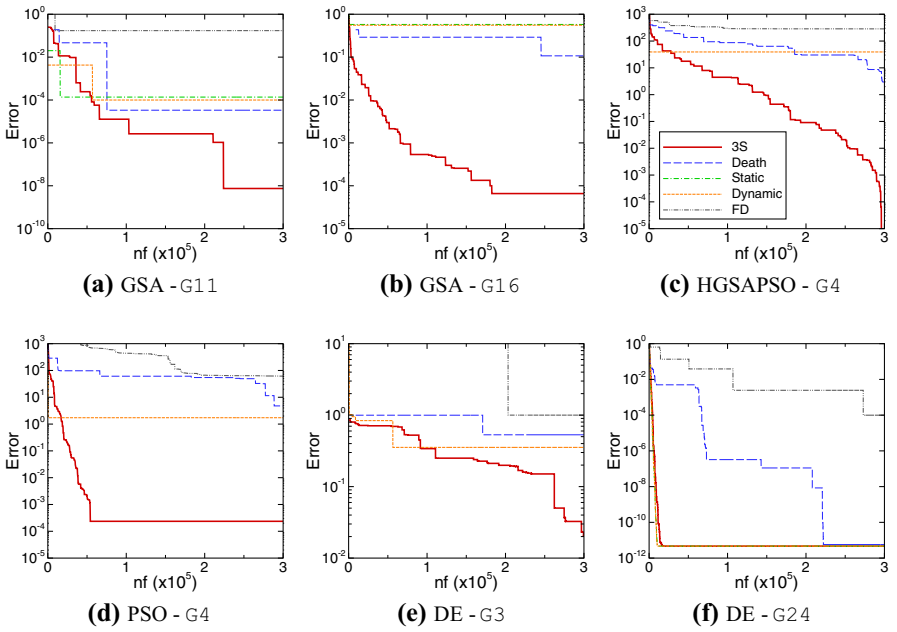
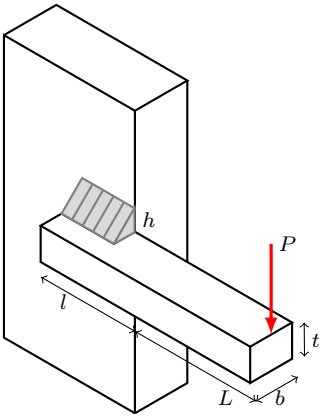


Fig. 2 Convergence history examples

Table 7 Summary of engineering benchmark problems

Problem	d	x_1	x_2	x_3	x_4	G
Welded beam	4	h	l	t	b	7
Pressure vessel	4	T_s	T_h	R	L	4
Spring	3	N	D	d	–	4

Fig. 3 Welded beam design problem



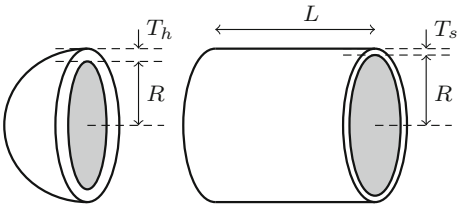


Fig. 4 Pressure vessel design geometry

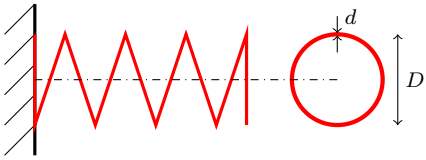


Fig. 5 Spring design geometry

Table 8 Results of welded beam design

	$f(\mathbf{x})$				$n(g_k \leq e)$		
	Best	Median	μ	σ	$e = 0$	10^{-6}	10^{-3}
GSA							
3S	1.727402	1.729828	1.730255	0.001801	0	0	1
Death	1.808322	1.993245	2.003017	0.079434	0	0	0
Static	1.973101	2.181876	2.217732	0.129108	0	0	0
Dyn	1.955687	2.214709	2.241077	0.125929	0	0	0
FD	2.475954	3.973515	4.050357	0.660359	0	0	0
PSO							
3S	1.724852	1.724852	1.724852	0.000000	4	4	4
Death	1.745077	1.782879	1.786890	0.023621	0	0	0
Static	1.724852	1.724852	1.724852	0.000000	4	4	4
Dyn	1.724852	1.724852	1.724852	0.000000	4	4	4
FD	INF	INF	INF	INF	–	–	–
HGSAPSO							
3S	1.724852	1.724853	1.724853	0.000000	0	1	2
Death	1.732127	1.749601	1.752745	0.012317	0	0	0
Static	1.724852	1.724852	1.724857	0.000013	0	1	3
Dyn	1.724852	1.724852	1.724854	0.000002	0	1	1
FD	INF	INF	INF	INF	–	–	–
DE							
3S	1.724852	1.724852	1.724852	0.000000	4	4	4
Death	1.726051	1.748840	1.769899	0.045957	0	0	0
Static	1.725929	1.732104	1.733871	0.007032	0	0	0
Dyn	1.728369	1.734565	1.736856	0.007460	0	0	0
FD	1.863230	2.145507	2.163205	0.150832	0	0	0

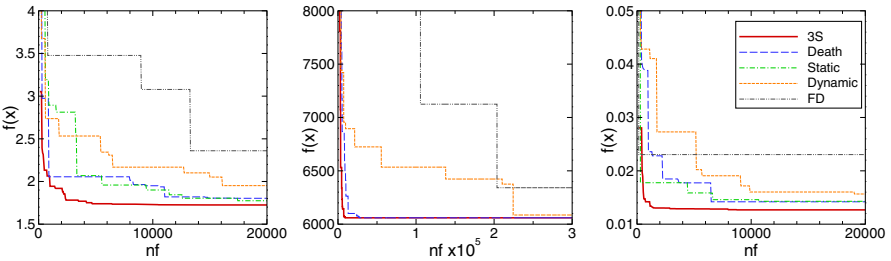
Table 9 Results of pressure vessel design

	$f(\mathbf{x})$				$n(g_k \leq e)$		
	Best	Median	μ	σ	$e = 0$	10^{-6}	10^{-3}
GSA							
3S	6090.5612	6410.7355	6469.1849	290.7843	0	0	1
Death	25231.964	79484.325	97988.687	55215.75	0	0	0
Static	INF	INF	INF	INF	–	–	–
Dyn	7048.1441	8498.6898	9284.6751	2022.200	0	0	0
FD	6607.3304	44975.829	64335.835	53664.30	0	0	0
PSO							
3S	6059.7143	6370.7797	6354.62534	301.678	2	2	2
Death	6322.2752	7130.7608	7134.94133	426.187	0	0	0
Static	INF	INF	INF	INF	–	–	–
Dyn	INF	INF	INF	INF	–	–	–
FD	6094.3048	6708.4793	6742.0074	364.0190	0	0	0
HGSAPSO							
3S	6059.7143	6090.5262	6284.8926	303.2683	0	1	1
Death	6086.4761	6766.5941	6826.3615	429.5686	0	0	0
Static	INF	INF	INF	INF	–	–	–
Dyn	INF	INF	INF	INF	–	–	–
FD	6101.0598	6540.8540	6599.2993	269.6939	0	0	0
DE							
3S	6059.7143	6059.7143	6059.7143	0.000000	2	2	2
Death	6059.7143	6059.7143	6061.7982	7.194272	0	0	1
Static	INF	INF	INF	INF	–	–	–
Dyn	6085.9670	6166.4935	6183.1322	72.07499	0	0	1
FD	6341.6718	7259.1855	7462.1871	617.9285	0	0	0

Figure 6b shows the convergence history of the five constraint handling methods tested when used with DE. The full convergence history is shown (up to the maximum number of allowed function evaluations) to fully compared the constraint handling methods. As in the welded beam problem, the convergence in the pressure vessel problem of the 3S framework is very quick, converging within 10,000 function evaluations. Again, this demonstrates that using the 3S framework does not restrict the good convergence properties of DE, but instead promotes them. Figure 6b also shows that while other commonly used constraint handling frameworks (such as the dynamic penalty and feasible directions) produce final answers that are comparable to the results obtained using 3S, the convergence of these constraint handling methods is relatively poor.

Table 10 Results of spring design

	$f(\mathbf{x})$				$n(g_k \leq e)$		
	Best	Median	μ	σ	$e = 0$	10^{-6}	10^{-3}
GSA							
3S	0.0126681	0.0127075	0.0127198	0.00004490	0	0	2
Death	0.0131882	0.0139745	0.0141306	0.00068372	0	0	0
Static	0.0131824	0.0151957	0.0152585	0.00091868	0	0	0
Dyn	0.0128781	0.0145724	0.0147026	0.00118987	0	0	0
FD	0.0201481	0.0298590	0.0394138	0.02027066	0	0	0
PSO							
3S	0.0126677	0.0127658	0.0128040	0.00011999	0	2	2
Death	0.0128709	0.0132964	0.0132885	0.00021451	0	0	1
Static	0.0126678	0.0127683	0.0128075	0.00012572	2	2	2
Dyn	0.0126683	0.0127873	0.0128325	0.00014446	2	2	2
FD	0.0137871	0.0137871	0.0144568	0.00066971	1	1	2
HGSAPSO							
3S	0.0126653	0.0127117	0.0127156	0.00003998	0	2	2
Death	0.0128331	0.0131382	0.0132394	0.00033515	0	0	0
Static	0.0126664	0.0127460	0.0129018	0.00031763	0	2	2
Dyn	0.0126704	0.0128376	0.0129157	0.00024546	0	2	2
FD	0.0127393	0.0136457	0.0141881	0.00132312	0	1	2
DE							
3S	0.0126652	0.0126653	0.0126652	0.00000019	2	2	2
Death	0.0127979	0.0132613	0.0133929	0.00030517	0	0	1
Static	0.0127307	0.0129435	0.0129736	0.00015238	0	0	0
Dyn	0.0127561	0.0129803	0.0130677	0.00021482	0	0	1
FD	0.0134464	0.0161174	0.0163794	0.00184867	0	0	0



(a) Welded beam **(b)** Pressure vessel **(c)** Spring

Fig. 6 Convergence histories of engineering benchmark problems using DE

Table 11 Comparison of statistical results in previous publications with 3S-DE

Ref.	Welded Beam		Pressure Vessel		Spring	
	Best	Mean	Best	Mean	Best	Mean
1	1.748309	–	6288.7445	–	0.0127047	–
2	1.724852	1.777692	6059.7016	6379.9380	0.012689	0.013165
3	1.92199	2.83721	6544.27	9032.55	0.0131200	0.0229478
4	1.724852	1.749040	6059.7143	6099.9323	0.0126652	0.0127072
5	1.724852	1.724852	6059.7143	6059.7277	0.0126652	0.0126652
6	1.724852	1.741913	6059.7147	6245.3081	0.012665	0.012709
7	1.724852	1.724852	6059.7143	6192.1162	0.012665	0.012683
8	1.724853	1.724853	–	–	0.012665	0.012713
9	–	–	6059.714	6410.087	0.012665	0.013165
10	1.724852	1.724852	6059.7143	6064.3360	0.0126653	0.0126770
11	1.724852	1.724852	6059.7143	6059.7143	0.0126652	0.0126652
Current	1.724852	1.724852	6059.7143	6059.7143	0.0126652	0.0126652

6.3 Spring design

The final results are shown in Table 10. In this design problem, the 3S framework produces optimization results that are always lower than every other constraint handling framework tested with all four of the swarm algorithms used. The median and mean solution also mirrors this result. Finally, the standard deviation is consistently small. These results further demonstrate the high performance of 3S both in terms of ability to find a low optimum solution and ability to robustly find that solution over multiple runs.

A convergence history for the first 20,000 function evaluations is shown in Table 6c. This demonstrates, as in the other engineering benchmark cases tested, the high convergence of a swarm algorithm with 3S used to handle the constraints. A solution that is close of the optimum is found within as few as 2000 function evaluations, and a converged solution is obtained within 10,000 function evaluations. The 3S method allows both a better solution to be obtained, as well as obtaining it with a quicker convergence rate.

6.4 Comparison to previously published results

To obtain a comparison of how the 3S method is working, as well as comparing to the other constraint handling methods outlined in this paper, previously published results are also given. The previously published results all represent good optimization results, with low optimum solutions. It should be noted that, unlike the CEC2006 functions, the optimum location of these engineering design problems is not known exactly. It therefore makes sense to compare to other results previously published. The results compared to are (1) Coello Coello (2000); (2) Mezura-

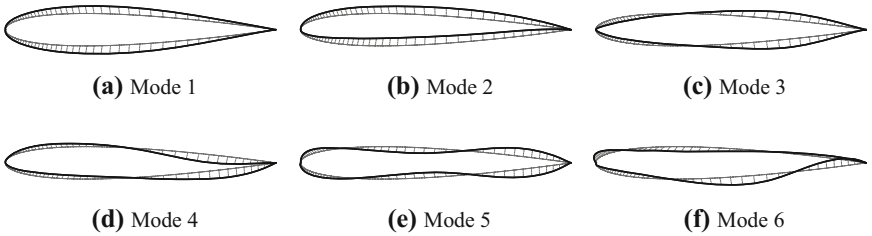


Fig. 7 First six aerofoil modes

Montes and Coello Coello (2005); (3) Parsopoulos and Vrahatis (2005); (4) He and Wang (2007); (5) Kim et al. (2010); (6) Akay and Karaboga (2012); (7) Brajevic and Tuba (2013); (8) Sadollah et al. (2013); (9) Gandomi (2014); (10) Baykasoglu and Ozsoydan (2015); (11) Salimi (2015). These represent a spread in the most competitive results obtained for these benchmarks using both historically successful algorithms and more recent state-of-the-art algorithms. For all of the engineering benchmark problems tested, the 3S framework combined with the differential evolution swarm optimizer produces the best solution with the lowest objective function (compared to using GSA, PSO or HGSAPSO). Hence, the solutions found using the this combination are compared to the previously published results, and this is outlined in Table 11.

The comparison to the previously published results demonstrate, and emphasise the good performance of the 3S framework for handling constraints. The results compare well, and often outperform other results previously published. A particularly encouraging result is that the mean value of the runs performed using the 3S-DE algorithm is lower than all other results highlighted in all of the benchmark functions tested. The best solution found is always either lower or at least as good as all of the results highlighted; an important result.

7 Aerodynamic design example

The performance of the new constrained optimization framework has been successfully demonstrated for a suite of analytical test functions and engineering benchmark functions. The final aspect of the work presented in this paper is demonstration of the framework within a typical ASO process, which is presented in this section.

ASO is the process of numerically solving the optimum design problem for aerodynamic bodies at specific flow conditions. Due to the high cost of the objective function evaluation, which is a CFD solution, the choice of optimization algorithm becomes important, and often, performing a low number of full CFD solutions is the primary driver for choosing a specific class of algorithm. Hence, agent-based algorithms tend not to be chosen, and instead gradient-based methods are employed. However, in a number of independent studies, multiple local optimum solutions for ASO problems have been demonstrated (Namgoong et al. 2002; Khurana et al.

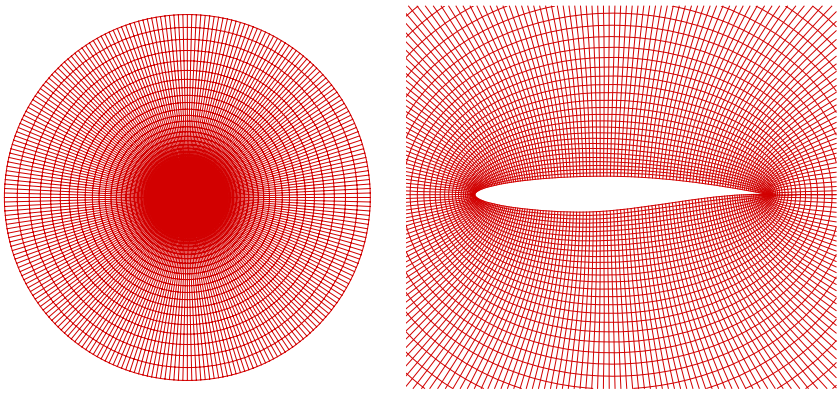


Fig. 8 257 × 97 O-mesh for RAE2822 aerofoil

2010; Leung and Zingg 2012), so the use of agent-based algorithms is important for performing aerodynamic design. As such, ASO is a suitable application area to consider the 3S constraint handling framework.

7.1 Geometry and mesh control

The shape parameterization and mesh deformation module must be flexible enough to allow sufficient design space investigation, robust enough to be applicable to any geometry, and efficient enough to maximise design space coverage with a minimum number of design parameters. The integration of global search algorithms into the aerodynamic shape optimization process necessitates the requirement for a minimum number of design variables to reduce computational burden as much as possible. An efficient aerodynamic optimization process is obtained by using a singular value decomposition (SVD) approach (Poole et al. 2015b) to derive an efficient, orthogonal set of design parameters. These are obtained by decomposing a set of 100 training aerofoils which all have high transonic performance to find the aerofoil mode shapes. The mode shapes are then used to deform the aerofoil surface during the optimization.

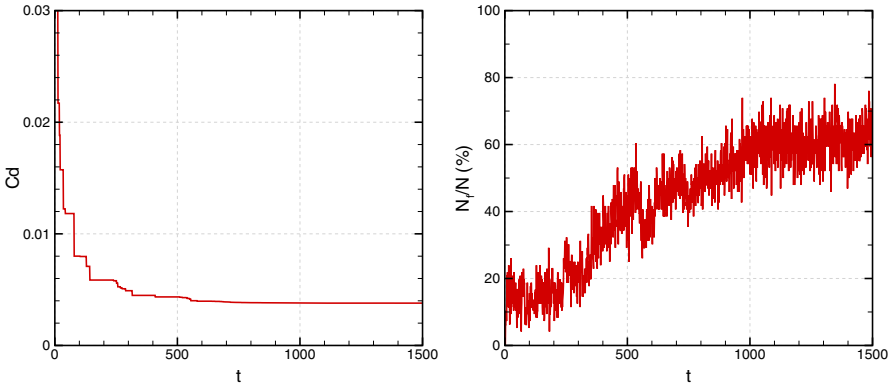
Using the SVD modes, a new shape is constructed by deforming a current aerofoil shape using a combination of the modes, as given by Eq. 17 where $\mathbf{X}^{new}$ and $\mathbf{X}^{initial}$ are the deformed surface and initial (undeformed) surface respectively. The mth modal deformation is given by the mth column of the modal matrix, $\mathbf{U}$, shown in Fig. 7. The first D columns of the modal matrix are used, and the design variables are the magnitude of the deformations, $\boldsymbol{\beta}$.

$$\mathbf{X}^{new} = \mathbf{X}^{initial} + \sum_{m=1}^{D} \beta_m \mathbf{U}_m \tag{17}$$

Deformations of the surface are propagated through the full CFD volume mesh using a set of 24 control points that are on the surface. These are linked to the CFD volume mesh using radial basis functions (RBF), wherein global interpolation is

Table 12 Aerodynamic optimization results (C_d in counts)

Design variables	C_l	C_d	$\Delta C_d(\%)$	C_m	V
Initial	1.02	199.8	–	−0.145	0.0779
6 modes + pitch	1.02	42.6	−78.7	−0.145	0.113
8 modes + pitch	1.02	38.4	−80.7	−0.139	0.0832
10 modes + pitch	1.02	37.9	−81.0	−0.128	0.116



(a) Objective **(b)** Feasible agents

Fig. 9 Convergence history of objective and percentage of feasible agents through optimization

used to provide direct control of the design surface and the CFD mesh, which is deformed in a high-quality fashion (Rendall and Allen 2008).

7.2 Flow-solver

The flow-solver used for objective function evaluation is a structured multiblock finite-volume, inviscid upwind code using the flux vector splitting of van Leer (1982). Convergence acceleration is achieved through multigrid (Allen 2002). A high-quality single block O-mesh was generated using a conformal mapping approach, and Fig. 8 shows two views of the 257×97 point mesh used for the test case, which extends to 100 chords at farfield. All surface cells have an aspect ratio of one.

7.3 Design problem

The case tested is the drag reduction of the RAE2822 aerofoil at $M = 0.73$, $\alpha = 2.67°$. This is a highly loaded case with $C_l = 1.02$. The flow is governed by the Euler equations, hence is an inviscid optimization and the primary drag component is the wave drag. The problem is constrained by the lift, moment and internal volume of the aerofoil. Hence, the problem can be written as:

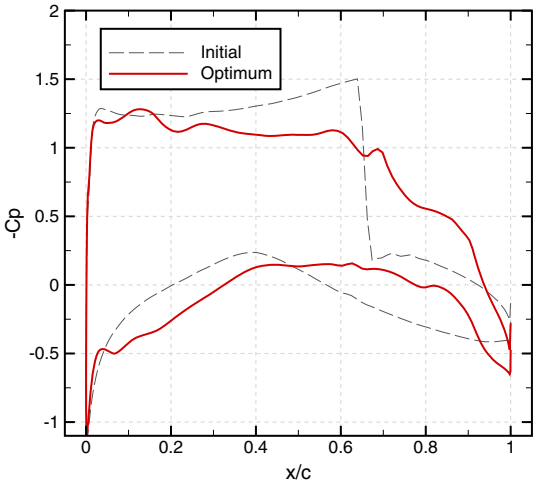


Fig. 10 Surface pressure coefficients

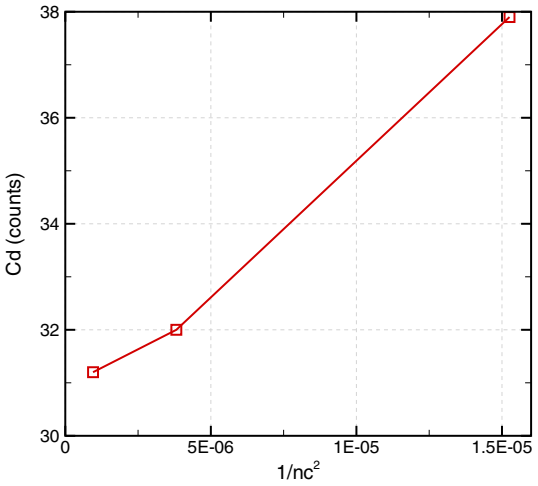


Fig. 11 Mesh convergence

Objective :	Minimize drag(C_D) with respect to modal deformation magnitude
Constraint 1 (lift) :	$C_l \geq C_l^{initial}$
Constraint 2 (pitching moment) :	$\lvert C_m \rvert \leq \lvert C_m^{initial} \rvert$
Constraint 3 (internal volume) :	$V \geq V^{initial}$

A similar optimization problem, albeit for viscous flow, is currently being studied for the AIAA Aerodynamic Design Optimization Discussion Group[1]; see Telidetzki et al. (2014), Bisson et al. (2014), LeDoux et al. (2015), Poole et al. (2015a) for example results. While the cost of the objective function evaluation for viscous flow

[1] https://info.aiaa.org/tac/ASG/APATC/AeroDesignOpt-DG/default.aspx.

is greater than inviscid flow, shocks in equivalent inviscid flows tend to be stronger and less smeared, making it a difficult optimization problem.

The design variables used were the first 6, 8 and 10 modes from the SVD method. A pitch design variable was also included to allow load balancing.

7.4 Results

The final drag results obtained for each number of design variables are given in Table 12. The convergence of the objective function and the percentage of feasible agents through the optimization is shown in Fig. 9; these are for the 10 mode case. It can be seen that due to using orthogonal design variables, the design space of N design variables is always contained within $N + n$ design variables, hence the ideal results for optimization mean that:

$$J(N + n) \leq J(N)$$

This is the trend that is seen. The monotonically decreasing final optimized drag results with increasing design variables indicates the success at finding a global optimum for this highly loaded aerodynamic problem. Furthermore, it is clear that for all three of the design variables tested, the final result lies on the lift constraint, as expected. It is interesting to note, however, that the lift constraint is often the only active constraint, and that the volume of the optimized aerofoils is always greater than the initial aerofoil. The moment constraint is only active on the six design variable case. The fact that the constraints are active indicates that it is likely that even towards the end of the optimization, a good number of agents will be infeasible, and this is shown in Fig. 9. This also shows that the agents spend a good amount of time at the start of the optimization searching through the infeasible space for good solutions, indicating the exploration ability of the approach. Towards the end of the optimization, exploitation becomes key and the number of feasible agent increases.

The surface pressure coefficient distributions of the initial and best optimized solution (10 design variable case) are shown in Fig. 10. These show that the constrained global search framework has successfully eliminated the shock, and therefore shows the largest drag reduction. Finally, a mesh convergence study has been performed with the deformed mesh from the 10 mode optimization results used as the basis and presented in Fig. 11. This mesh has been doubled in each direction twice, and indicates that the final mesh converged solution is slightly less than 31 drag counts.

8 Concluding remarks

A new, generic framework for handling constraints when performing constrained optimization using agent-based search algorithms has been presented. The constraint handling framework is called separation-sub-swarm (3S) and has the advantage of being a high performance framework that is independent of the type of

swarm algorithm used for an optimization, so can be added to the general swarm algorithm system. The new method works by considering the whole population of agents as two independent sub-swarms. All feasible agents optimize the objective function value, where the user selected swarm algorithm acts on this, the infeasible agents then minimise the constraint violation. This independence of the infeasible swarm from the feasible swarm allows the 3S framework to be added on any existing swarm algorithm, including those algorithms where the fitness function value is used within the agent update scheme.

Constrained analytical tests on the CEC2006 suite of analytical benchmark functions and on three standard engineering design problems have been presented using the 3S framework coupled to a particle swarm optimization (PSO), gravitational search algorithm (GSA), hybrid-GSA-PSO and differential evolution (DE). The 3S framework was compared to a death penalty, a static penalty, a self adaptive dynamic penalty and a feasible directions approach. Results showed that overall, the 3S framework produced solutions that were much more likely to be feasible, returning average feasibility rates of over 90% for all swarm algorithms tested. Results were, in general, closer to the theoretical best solution available compared to the other frameworks tested. For the engineering benchmarks, the framework produced final optimum solutions that are at least as good (and often better) than the results published previously. This demonstrates the high performance of the 3S framework within the umbrella of constraint handling frameworks.

Finally, the constraint handling framework was applied to an aerodynamic shape optimization problem for transonic drag minimization of an aerofoil. The design variables used were obtained using a modal extraction technique the produces orthogonal design variables. Results demonstrate that the new framework is capable of producing shock-free optimization results in inviscid flow. Furthermore, the use of orthogonal design variables should result in improved optimization performance for an increasing number of design variables, and this is demonstrated indicating that the constrained global method has successfully located the global optimum solution.

References

Akay B, Karaboga D (2012) Artificial bee colony algorithm for large-scale problems and engineering design optimization. J Intell Manuf 23(4):1001–1014

Allen CB (2002) Multigrid convergence of inviscid fixed- and rotary-wing flows. Int J Numer Meth Fluids 39(2):121–140

Allen CB, Rendall TCS (2013) Computational-fluid-dynamics-based optimisation of hovering rotors using radial basis functions for shape parameterisation and mesh deformation. Optim Eng 14:97–118

Baykasoglu A, Ozsoydan FB (2015) Adaptive firefly algorithm with chaos for mechanical design optimization problems. Appl Soft Comput 36:152–164
Belegundu AD, Arora JS (1985) A study of mathematical programming methods for structural optimization. Part II: numerical results. Int J Numer Methods Eng 21(9):1601–1623
Bisson F, Nadarajah SK, Shi-Dong D (2014) Adjoint-based aerodynamic optimization framework. In: 52nd AIAA aerospace sciences meeting. National Harbor, Maryland, AIAA paper 2014-0412
Brajevic I, Tuba M (2013) An upgraded artificial bee colony (ABC) algorithm for constrained optimization problems. J Intell Manuf 24(4):729–740
Clerc M (2013) Particle swarm optimization. Wiley-ISTE, Hoboken
Coello Coello CA (2000) Use of a self-adaptive penalty approach for engineering optimization problems. Comput Ind 41(2):113–127
Das S, Suganthan PN (2011) Differential evolution: a survey of the state-of-the-art. IEEE Trans Evol Comput 15(1):4–31
Deb K (2000) An efficient constraint handling method for genetic algorithms. Comput Methods Appl Mech Eng 186:311–338
Engelbrecht AP (2005) Fundamentals of computational swarm intelligence. Wiley, Hoboken
Gandomi AH (2014) Interior search algorithm (ISA): a novel approach for global optimization. ISA Trans 53:1168–1183
He Q, Wang L (2007) A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization. Appl Math Comput 186(2):1407–1422
Hicken JE, Zingg DW (2010) Aerodynamic optimization algorithm with integrated geometry parameterization and mesh movement. AIAA J 48(2):400–413
Hu X, Eberhart R (2002) Solving constrained nonlinear optimization problems with particle swarm optimization. In: 6th world multiconference on systemics, cybernetics and informatics (SCI 2002), Orlando, Florida
Jameson A, Pierce NA, Martinelli L (1998) Optimum aerodynamic design using the Navier–Stokes equations. Theor Comput Fluid Dyn 10(1):213–237
Kannan BK, Kramer SN (1994) An augmented lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design. J Mech Des 116(2):405–411
Kennedy J, Eberhart R (1995) Particle swarm optimization. In: 1995 IEEE international conference on neural networks, Perth, Australia
Khurana MS, Winarto H, Sinha AK (2010) Airfoil optimisation by swarm algorithm with mutation and artificial neural networks. In: 47th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition, Orlando, Florida, AIAA paper 2009-1278
Kim TH, Maruta I, Sugie T (2010) A simple and efficient constrained particle swarm optimization and its application to engineering design problems. Proc Inst Mech Eng C J Mech Eng Sci 224(2):389–400
LeDoux ST, Vassberg JC, Young DP, Fugal S, Kamenetskiy D, Huffman WP, Melvin RG, Smith MF (2015) Study based on the AIAA aerodynamic design optimization discussion group test cases. AIAA J 53(7):1910–1935
Leung TM, Zingg DW (2012) Aerodynamic shape optimization of wings using a parallel newton-krylov approach. AIAA J 50(3):540–550
Liang JJ, Suganthan PN (2006) Dynamic multi-swarm particle swarm optimizer with a novel constraint-handling mechanism. In: 2006 IEEE congress on evolutionary computation, Vancouver, Canada
Liang JJ, Runarsson TP, Mezura-Montes E, Clerc M, Suganthan PN, Coello Coello CA, Deb K (2013) Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization. Technical report, Nanyang Technological University, Singapore
Lu H, Chen W (2006) Dynamic-objective particle swarm optimization for constrained optimization problems. J Comb Optim 12(4):409–419
Lu H, Chen W (2008) Self-adaptive velocity particle swarm optimization for solving constrained optimization problems. J Glob Optim 41(3):427–445
Martins JRRA, Alonso JJ, Reuther JJ (2004) High-fidelity aerostructural design optimization of a supersonic business jet. J Aircr 41(3):523–530
Mezura-Montes E, Coello Coello CA (2005) Useful infeasible solutions in engineering optimization with evolutionary algorithms. In: 4th Mexican international conference on artificial intelligence, published in MICAI 2005: advances in artificial intelligence
Mezura-Montes E, Coello Coello CA (2011) Constraint handling in nature-inspired numerical optimization: past, present and future. Swarm Evol Comput 1:173–194

Mirjalili S, Hashim SZM, Sardroudi HM (2012) Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm. Appl Math Comput 218(22):11,125–11,137

Namgoong H, Crossley W, Lyrintzis AS (2002) Global optimization issues for transonic airfoil design. In: 9th AIAA/ISSMO symposium on multidisciplinary analysis and optimization, Atlanta, Georgia, AIAA Paper 2002-5641

Parsopoulos KE, Vrahatis MN (2002) Particle swarm optimization method for constrained optimization problems. Euro Int Symp Comput Intell 2002:214–220

Parsopoulos KE, Vrahatis MN (2005) Advances in natural computation. In: Wang L, Chen K, Ong YS (eds) Unified particle swarm optimization for solving constrained engineering optimization problems. Springer, Berlin, pp 582–591

Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization: an overview. Swarm Intell 1(1):33–57

Poole DJ, Allen CB, Rendall TCS (2015a) Control point-based aerodynamic shape optimization applied to AIAA ADODG test cases. In: 53rd AIAA aerospace sciences meeting, Kissimmee, Florida, AIAA Paper 2015-1947

Poole DJ, Allen CB, Rendall TCS (2015b) Metric-based mathematical derivation of efficient airfoil design variables. AIAA J 53(5):1349–1361

Rao SS (2013) Engineering optimization: theory and practice, 3rd edn. New Age International Publishers, New Delhi

Rashedi E, Nezamabadi-pour H, Saryazdi S (2009) GSA: a gravitational search algorithm. Inf Sci 179:2232–2248

Rendall TCS, Allen CB (2008) Unified fluid–structure interpolation and mesh motion using radial basis functions. Int J Numer Methods Eng 74(10):1519–1559

Sadollah A, Bahreininejad A, Eskandar H, Hamdi M (2013) Mine blast algorithm: a new population based algorithm for solving constrained engineering optimization problems. Appl Soft Comput 13(5):2592–2612

Salimi H (2015) Stochastic fractal search: a powerful metaheuristic algorithm. Knowl Based Syst 75:1–18

Storn R, Price K (1995) Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, ICSI, UC Berkeley, tR-95-012

Sun CL, Zeng JC, Pan JS (2011) An improved vector particle swarm optimization for constrained optimization problems. Inf Sci 181:1153–1163

Takahama T, Sakai S (2005a) Constrained optimization by the alpha constrained particle swarm optimizer. J Adv Comput Intell Intell Inf 9(3):282–289

Takahama T, Sakai S (2005b) Constrained optimization by the epsilon constrained particle swarm optimizer with epsilon-level control. Adv Soft Comput 29:1019–1029

Telidetzki K, Osusky L, Zingg DW (2014) Application of jetstream to a suite of aerodynamic shape optimization problems. In: 52nd AIAA aerospace sciences meeting, National Harbor, Maryland, AIAA paper 2014-0571

Tsai HC, Tyan YY, Wu YW, Lin YH (2013) Gravitational particle swarm. Appl Math Comput 219(17):9106–9117

Vanderplaats GN (1999) Numerical optimization techniques for engineering design, 3rd edn. Vanderplaats Research and Development Inc, Monterey

van Leer B (1982) Flux-vector splitting for the Euler equations. In: Eighth international conference on numerical methods in fluid dynamics. Lecture notes in physics, pp 507–512

Venter G, Sobieszczanski-Sobieski J (2003) Particle swarm optimization. AIAA J 41(8):1583–1589